

Ex. 10, Simulink

Stefan Karlsson

May 14, 2014

1 Introduction

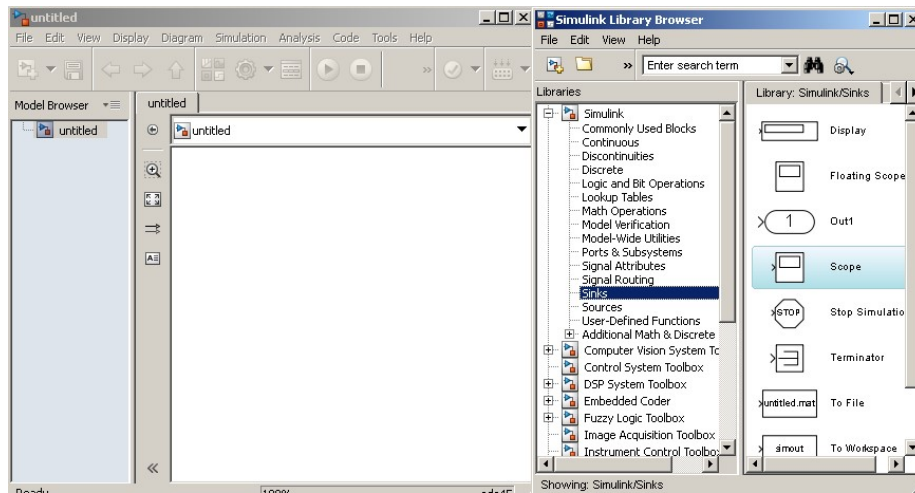



Figure 1: Appearance of simulink

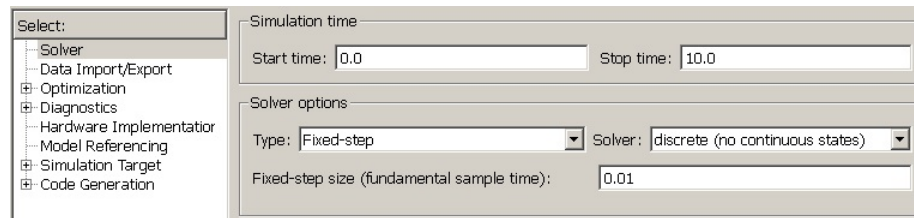
In this exercise we will get familiar with Simulink (version 8.1.). Simulink is the biggest “side-product” to Matlab. It is a block-based application that simulates systems. It sees a lot of use in systems modeling and engineering. There is a huge amount of applications of Simulink, with alot of extra functionalities and extensions. This exercise will give you only a very basic introduction. Now, to get started:

```
simulink;
```

There will be a number of windows opening. One will be Simulink Library Browser. There we can see some of the sublibraries with different applications including Control Theory, Fuzzy logic, Signal Processing, Image Processing and Telecommunication. Create a simulink model file(CTRL+N). Have both windows open (Simulink Library Browser and model file) without any overlap looking something like figure 1.

1.1 Setup Model Configuration

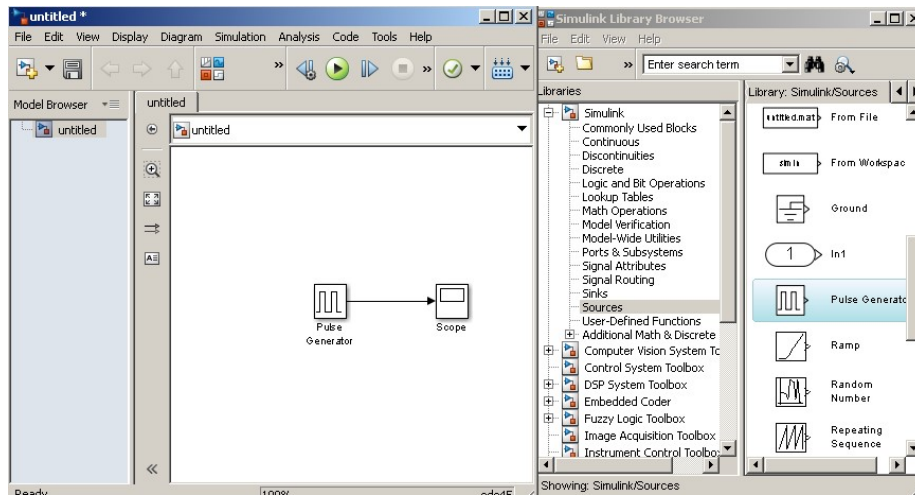
In the toolbar of the simulink model view, click the icon , or click **Model Configuration Parameters** in the menu. There will be a settings dialog, chose the option **Solver**. In **Solver Options**, change from "Variable-step" to "fixed-step". For option **Solver** chose "discrete", and for the sample time, chose 0.01. You should see the following when done:




With these settings, simulink will behave in a very predictive way (as we will see soon). However, for general system simulations, these are usually not the best settings.

2 Standard useful blocks

With the configuration settings of section 1.1, make a simulink model by the following steps. First, enter Simulink **Library Browser** → **Simulink** and click on **Sinks**. There is a symbol named **Scope** which is used to visualize signals being processed at any time in simulations. Click it, and drag and drop in your model view. Get a signal source. Enter **Simulink** → **Sources** and choose **Pulse Generator**, click and drag the symbol onto your model as before. The only thing that remains to be done is to connect the Pulse Generator with Scope. Press the left mouse button and drag from the arrow on the Pulse Generator to the input on the Scope. The result should look like:




We have a pulse generator connected to a scope, but how does the signal look like? Run the simulation by clicking the little black arrow icon  (or menu: **Simulation** → **Run**). After you have run it, you can double click the scope to see what the signal looks like. It seems to be a square wave pulse.

Lets set the period to 1/5 seconds, and amplitude to 2 of the pulse being generated. This is achieved by double clicking on the block for the pulse generation. Run the simulation again, double click on Scope to see how it looks like now. It seems to be rather many pulses. Use the tools menu of the scope figure to get a good view of the signal. There are ways to zoom, pan and scale the figure interactively.

Now, get a Signal Generator and connect it like previously:



Double click on it. Different signals in the Signal generator are available like square wave, sawtooth and sine wave. Choose a sine wave with the default parameters, then run the simulation. This gives you a nice sinus wave in the scope.

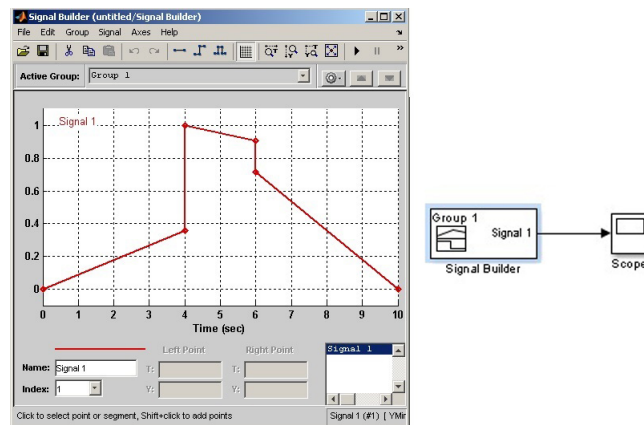
Now, change the amplitude of the wave to 2 and frequency to 100 Hz, and run the simulation again. The output in the scope should look weird, cluttered. That's because we dont have enough sampling frequency for our wave(our sampling time is too high). The sampling time in our simulation is called "step-size", and you can access it by the button , as in section 1.1. In solver options, change "Fixed-step size" to 0.0001, and change stop-time to 0.1 second. Run

the simulation again, there should now be a nice sinus, covering the simulation time 0-0.1 seconds.

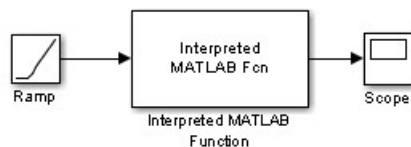
NOTE, that when we leave the model configurations in their default settings, Simulink will be able to modify the step size automatically to fit the problem. We will let Simulink do that most of the time (and not do the changes of section 1.1).

2.1 Defining signal sources

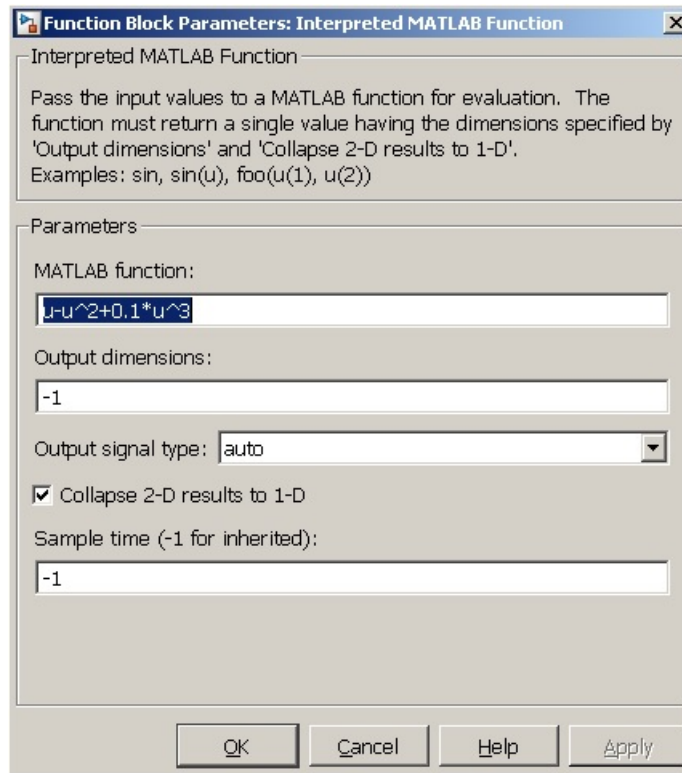
Make a new model that has default settings for model configurations (do NOT do the changes of section 1.1). In it add a "Signal Builder" block as a source. Open it and construct a signal like Figure below. Can be done by click and drag with the mouse button.



This allows you to custom-make signals graphically, but even more useful is to generate your own input signals is by creating a Matlab function. This can be done by getting **Interpreted Matlab Fcn** from the library **Simulink**→**User-defined Functions**. Put it together with a Scope in, choose **ramp** as input. This block is found in the directory **Simulink**→**Sources**→**Ramp**. The resulting model should look as follows:



Double clicking the block, you will see something like the following example:



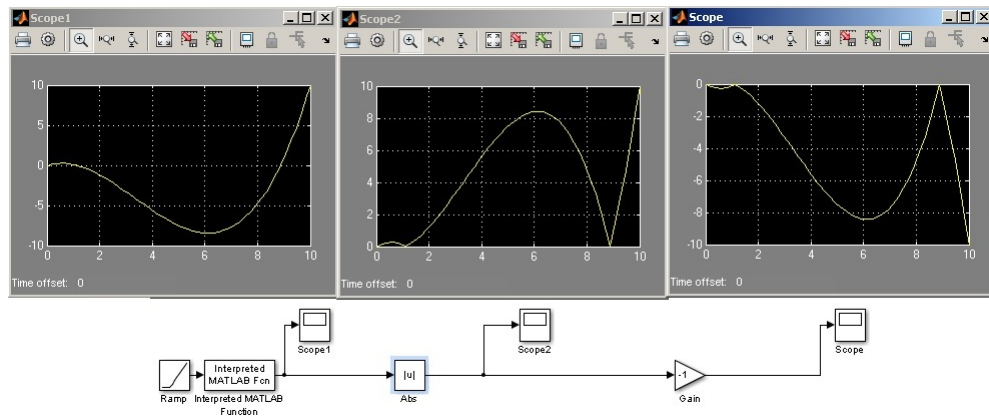
In the **MATLAB function** edit box you can write any matlab expression, and access your own function if you like (assuming that your function inputs 1 argument, and outputs 1). For now, try out the function $u - u^2 + 0.1u^3$

Click on the ramp source (a linearly increasing input). The block contains, slope, start time and initial output.

If we want to input a nice "linear time signal" into our function, set **slope**=1 and **start time**= 0 in our Ramp source. Then run the simulation. Double click the scope and see the output.

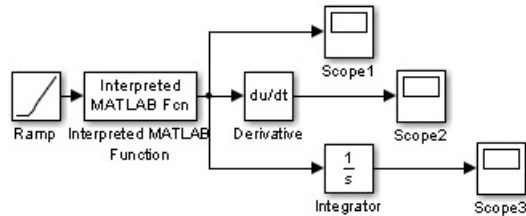
2.2 Math Blocks, routing and output

Extend the model from previous example by adding a **Gain** block and an **Abs** block (both found in the **Math Operations** sub-library), and construct it according to:



It should be clear that **Gain** multiplies the signal, and **abs** takes magnitude.

Next, use the **Derivative** and the **Integrator** blocks from the **Continuous** sub-library, to construct the following model:

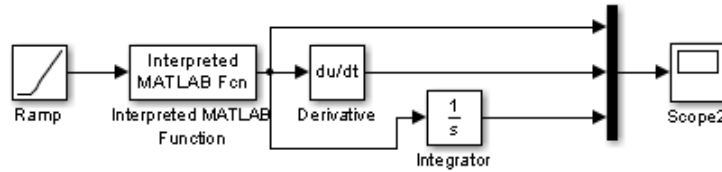


The symbol on the integrator block ($\frac{1}{s}$) may confuse you if you have never seen "transfer functions". We will not cover this topic in this course, so you can ignore the symbol on the block¹. From your perspective and background, a symbol like $(\int u)$ may seem more logical, and correspond to how you should consider what the block does: it accumulates value over time.

Now, Change the input signal (Matlab Function block in your model) to u^2 . The three scopes should contain the original signal, its derivative, and its integration respectively.

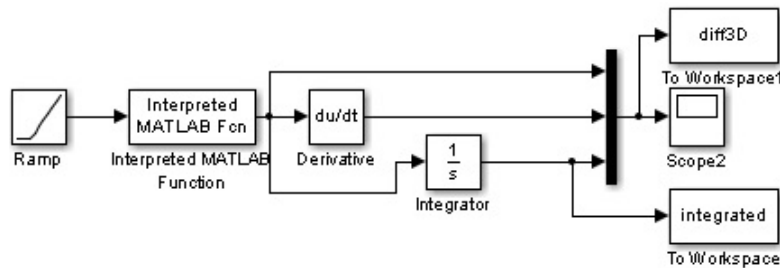
Next, lets see how we can merge links using a **Mux** block from **Signal Routing**. Add Mux block first, then double click it. In the field **Number of inputs** type in 3. You should notice that this updates the visual block in the model, to have 3 connectors on its left side. Update your model to look like this:

¹in the future, if you plan on studying linear systems or control theory, you would do well to remember this notation however



Notice that the single scope now takes three-signal input, which will be evident after you run your model and investigate the scope. Three graphs are apparent.

Next, let's export some data to the Matlab workspace. Add 2 blocks **Simout** from **Sinks**. In your new blocks, type in **variable name** `diff3D` and `integrated`, and connect them as follows:



After you run your model, there should be 3 new variables in your workspace: `diff3D`, `integrated` and `tout`. The variable `tout` is always generated when you run simulink models, and contains the time vector of the simulation². The variables `integrated` and `diff3D` contains the data we gathered, but are in a bit of a funky format. Here is how we get the relevant information:

```
%run this AFTER the simulink model that generates "diff3D" and "integrated"
t = integrated.Time;
sig = integrated.Data;
plot(t,sig);
legend('integrated');
```

similarly, we can get all three plots of the simulation:

```
t = diff3D.Time;
sig = diff3D.Data; %get all three signals
plot(t,sig);
legend('original','derivative','integrated');
```

²however, this vector is not necessarily the right size for your particular logged vector. Read on to see how to get the correct time vector for your logged data.

3 Differential Equations

Describing, analyzing and solving differential equations is where Simulink really shines. We will take two examples here, beginning with a simple model of bacterial growth and finishing off with a mechanics system.

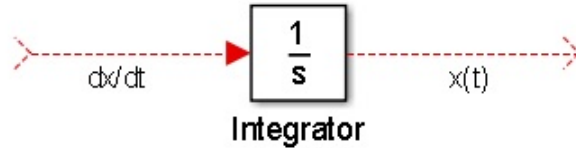
3.1 Bacterial Growth Model

Denote with $x(t)$ the amount of bacteria at time t . The bacterial growth ($\frac{dx(t)}{dt}$) is given by how many bacteria is born, offset by how many bacteria is dying. Our model has the growth of bacteria proportional to the number of current bacterias ($x(t)$), and the death of bacterias is proportional to the number squared ($x^2(t)$), thus:

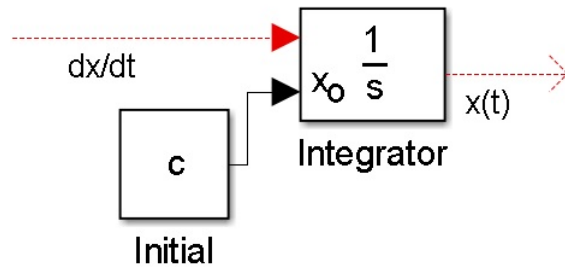
$$\begin{aligned} growthRate &= birthRate - deathRate \\ \frac{dx(t)}{dt} &= bx(t) - dx^2(t) \end{aligned}$$

Where the constants b and d are innate birth- and death constants respectively. Assume an initial value of $x(0) = c$ bacterias from the start.

There are many ways to describe this system in Simulink. A way that is particularly simple, starts by putting in an integrator:

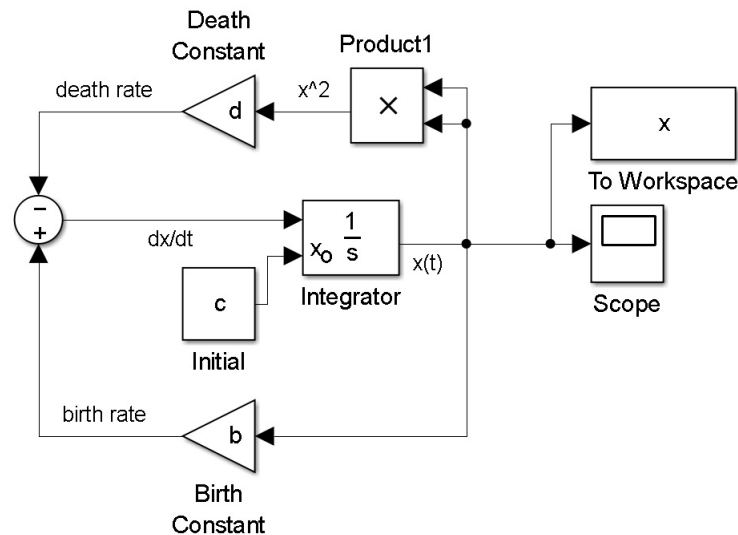


Notice the labels I put on the links, you can do that by double clicking links. Labels are like remarks in regular code: they do not affect any computation. An integrator can be thought of as an accumulator variable, that gets updated on each iteration. On the first iteration $t = 0$, there will be $x(0)$ being stored in the integrator. This is our initial condition. Double click the integrator, and find the option **initial condition source**, set it to external. This will add an input to the integrator block, onto which we put a constant:



Above we used the the **constant** block (in **commonly used blocks**) to attach to the integrator. Open the constant block, put **constant value** to **c**. Later, we will have to set **c** to a value.

In order to get a full system description from our current incomplete model, we have to **CLOSE THE LOOP**. From the right hand side of the integrator $x(t)$ we will perform operations through elementary blocks in order to get equality with $\frac{dx(t)}{dt}$, the left hand side. This will yield the finished system:



For this model, we used two new blocks: **Product** and **Sum**, and they are self-explanatory.

In order to flip and rotate blocks, simply right click them and find the options. Also, notice the important detail that the **sum** block has one input labeled **+**(plus), and one **-** (minus), making it in effect a "difference block".

In order to run the model, we need to set the parameters **b**, **d** and **c**. When running simulink, it automatically looks for variables in the workspace. There-

fore, the simplest way to set these constants is to run the simulink model from a script, using the `sim` function. Assuming you have saved above model as "bacterial":

```
%set the constants for the simulink model
b = 1;    %birth constant
d = 0.01; %death constant

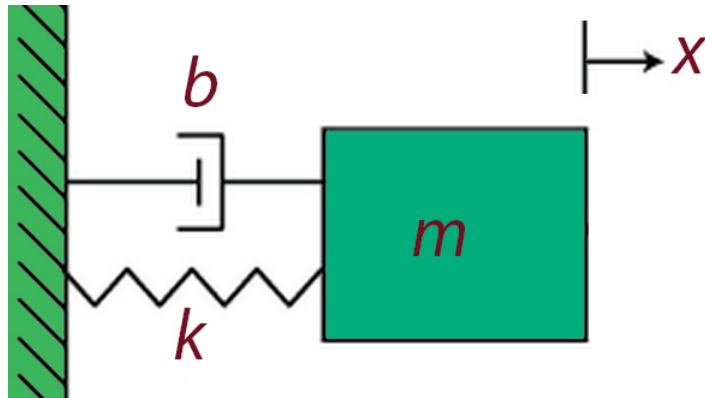
%set the initial conditions:
c = 2;    %initial volume of bacteria

%execute the simulink model(sets output variable 'x')
sim('bacterial');

% plot the result
plot(x.Time, x.Data);
```

3.2 Mass-Damper-Spring Mechanical System

Consider the following system:

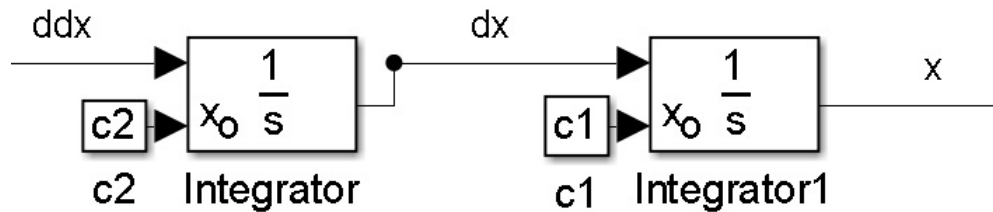


Assume that the spring is at equilibrium at $x = 0$. The total force on the mass ($F = m \frac{d^2x}{dt^2}$) is a sum of two sources, spring ($F_s = -kx$) and damper ($F_d = -p \frac{dx}{dt}$). We have:

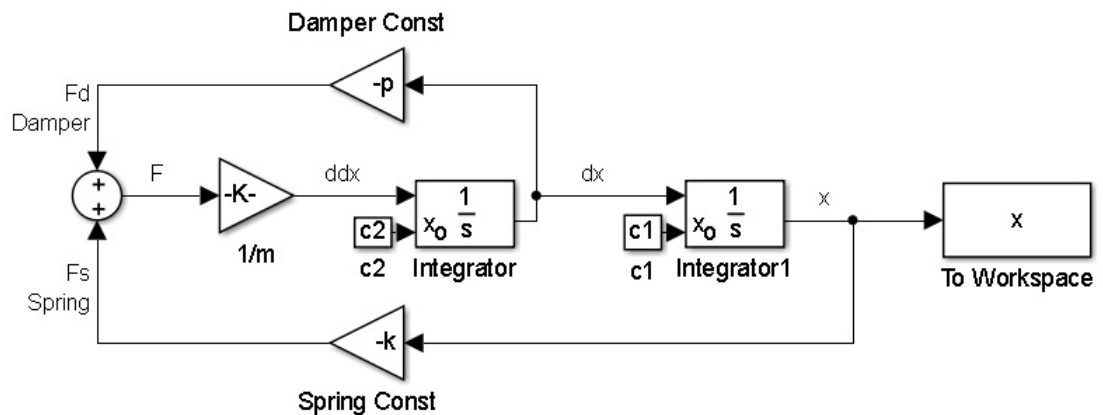
$$F = m \frac{d^2x}{dt^2} = -p \frac{dx}{dt} - kx$$

To solve this, we need the physical parameters (mass (m), damper constant (p) and spring constant (k)), as well as initial conditions ($x(0) = c_1, \dot{x}(0) = c_2$).

Like the bacterial example, let's formulate a basic relationship using integrators first, where we make initial conditions explicit:



In the basic incomplete model above, we now close the loops, and will end up with the following system:




Assuming above model is saved as "SpringDamp" we can run this using the script:

```
%set the constants for the simulink model
m = 5; %mass    [kg]
k = 2; %spring  [N / m]
p = 1; %damper  [Ns / m]

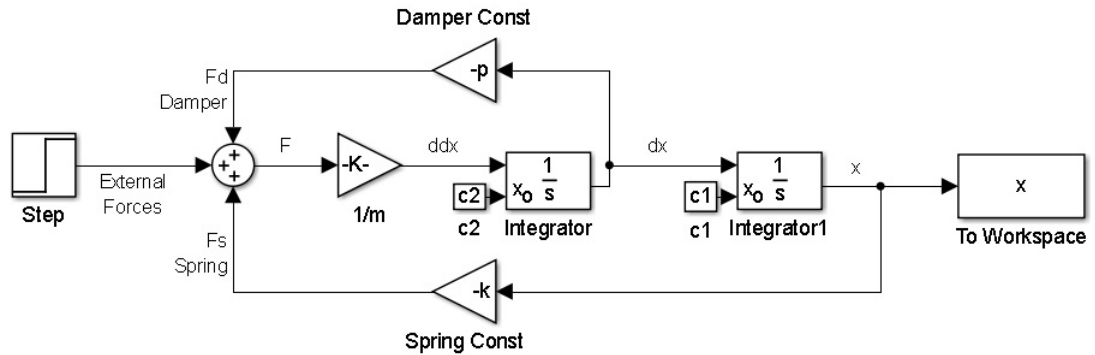
%set the initial conditions:
c1 = -10; %initial position
c2 = 0; %initial speed

%execute the simulink model(sets output variable 'x')
sim('SpringDamp');
plot(x.Time,x.Data);
```

For smooth visualization, we may want to change the default parameters of above model. Click the icon , or click **Model Configuration Parameters** in the menu. In **Solver**, change **Stop time** to 100, and **Max step size** from "auto" to 0.1. Then run the above script again, and hopefully get some smoother

curves.

To show the power of Simulink for analyzing differential equations, let's consider how easy it is to modify the system we have just made. Say that we wish to turn on gravity in the system, at exactly $t = 5$. If gravity acts, then there is an external force, and there are a total of 3 forces acting. From our model, we know that the place where forces are summed up are at the Sum block. Thus, we can easily include another source by modifying the model as follows:



The new thing in the model is the **Step** block, which is a signal source. It is set to start at 5 seconds in, and to go from 0 to 9.81 in value. Assuming this model is saved as "SpringDamp2", let's run it with the mass in equilibrium at start ($c_1 = c_2 = 0$) and see the behavior when gravity kicks in:

```
%set the constants for the simulink model
m = 5; %mass    [kg]
k = 2; %spring  [N / m]
p = 1; %damper  [Ns / m]

%set the initial conditions:
c1 = 0; %initial position
c2 = 0; %initial speed

%execute the simulink model(sets output variable 'x')
sim('SpringDamp2');
plot(x.Time,x.Data);
```

Tasks

Solutions that do not fill the following requirements **EXACTLY, PRECISELY AND TO THE LETTER** will not be considered:

- Send your solutions by email to:
stefan.karlsson@hh.se
subject: Matlab, Exercise **X**, **YourNames**
- Send all the files that are requested, no more and no less, in one single zip file per exercise, with NO sub-folders in the zip file. All the files, for all the tasks should be bundled into one zip file.
- Put the Names of the authors, in remarks, at the top of every m-file.
- Send the solutions within 2 weeks of every exercise session. That is, you have a two week deadline to hand it in.
- You will get 2 chances to send it in to me correctly.

Task 1

Create a simulink model `e10_1` that solves the differential equation:

$$\frac{dx}{dt} + x = a + \sin(bt)$$

with initial condition: $x(0) = 2$.

The model should be called from an m-file called `e10_1script.m` to start and one should also initiate the differential equation with constants from there. You must solve the differential equation within the simulink model, and export the solution to the matlab workspace. The solution must be plotted in sufficient detail and over a duration that makes sense.

Hand in 2 files, the simulink model `e10_1`, and the script file `e10_1script.m` (both should be in the zip file for this exercise).

Task 2 (Optional for grade 4)

A simulink model [e10_2](#) solves the following equation system consisting of 3 differential equations:

$$\begin{aligned}\dot{x}_1(t) &= x_1(t) + x_2(t) - 2x_3(t) \\ \dot{x}_2(t) &= 2x_1(t) - 2x_3(t) \\ \dot{x}_3(t) &= -2x_1(t) + 2x_2(t) + x_3(t)\end{aligned}$$

Assume the following initial conditions: $x_1(0) = 2$, $x_2(0) = 1$ and $x_3(0) = 1$

The model should be called from an m-file called [e10_2script.m](#) to start. You must solve the differential equation within the simulink model, and export the solution to the matlab workspace. The solution must be plotted in sufficient detail and over the duration 0-2 sec.

Hand in 2 files, the simulink model [e10_2](#), and the script file [e10_2script.m](#) (both should be in the zip file for this exercise).

Hints:

- This system of equations has been encountered in this course twice before.
- Expect the model to be bigger than what you have done in the exercise so far.